A Hierarchical Learning Scheme for Solving the Stochastic Point Location Problem

Anis Yazidi[1], Ole-Christoffer Granmo[1],
B. John Oommen[2,⋆], and Morten Goodwin

[1] Dept. of ICT, University of Agder, Grimstad, Norway
[2] School of Computer Science, Carleton University, Ottawa, Canada

Abstract. This paper deals with the Stochastic-Point Location (SPL) problem. It presents a solution which is novel in both philosophy and strategy to all the reported related learning algorithms. The SPL problem concerns the task of a Learning Mechanism attempting to locate a point on a line. The mechanism interacts with a random environment which essentially informs it, possibly erroneously, if the unknown parameter is on the left or the right of a given point which also is the current guess. The first pioneering work [6] on the SPL problem presented a solution which operates a *one-dimensional* controlled Random Walk (RW) in a discretized space to locate the unknown parameter. The primary drawback of the latter scheme is the fact that the steps made are always very conservative. If the step size is decreased the scheme yields a higher accuracy, but the convergence speed is correspondingly decreased.

In this paper we introduce the Hierarchical Stochastic Searching on the Line (HSSL) solution. The HSSL solution is shown to provide *orders of magnitude* faster convergence when compared to the original SPL solution reported in [6]. The heart of the HSSL strategy involves performing a controlled RW on a discretized space, which unlike the traditional RWs, is not structured on the line *per se*, but rather on a binary tree described by intervals on the line. The overall learning scheme is shown to be optimal if the effectiveness of the environment, p, is greater than the *golden ratio conjugate* [4] – which, in itself, is a very intriguing phenomenon. The solution has been both analytically analyzed and simulated, with extremely fascinating results. The strategy presented here can be utilized to determine the best parameter to be used in any optimization problem, and also in any application where the SPL can be applied [6].

Keywords: Stochastic-Point Problem, Discretized Learning, Learning Automata, Controlled Random Walk.

⋆ Chancellor's Professor; *Fellow: IEEE* and *Fellow: IAPR*. The Author also holds an Adjunct Professorship with the Dept. of ICT, University of Agder, Norway The author is grateful for the partial support provided by NSERC, the Natural Sciences and Engineering Research Council of Canada.

H. Jiang et al. (Eds.): IEA/AIE 2012, LNAI 7345, pp. 774–783, 2012.

1 Introduction

Learning algorithms which operate within a known domain (for example, within the unit interval), generally work by moving from their current location to a new point within its *immediate* neighborhood. The step size of the migration depends on the learning parameter, where a smaller step size leads to a slower but more accurate convergence, and conversely, a larger step size leads to a faster but less accurate convergence. In this paper, we propose a completely novel strategy regarding how one can move around within the domain of interest. Rather than consider the domain as a *single* interval, we model it is a sequence of increasingly larger intervals mapped onto an underlying Binary Search Tree (BST). Thus, effectively, we are operating a Random Walk (RW) on a BST, where the steps of the RW need not necessarily be in the close proximity of the current point. The most incredible facet of this strategy is that the RW converges to the underlying optimum point – even if the environment providing the information to the learning algorithm is faulty or erroneous.

The problem we investigate is Stochastic Point Location (SPL) [6] problem which can be described as follows. Consider the problem of a robot (algorithm, Learning Mechanism (LM)) moving along the real line attempting to locate a particular point λ^*. To assist the mechanism, we assume that it can communicate with an Environment ("Oracle"), which guides it with information regarding the direction in which it should go. If the Environment is deterministic, the problem is the "deterministic point location problem", which has been studied rather thoroughly. In an initial version, Baeza-Yates *et al.* [1] presented the problem in a setting such that the Environment could charge the robot a cost that is proportional to the distance it is from the point sought for. The question of having multiple communicating robots locate a point on the line has also been studied by Baeza-Yates *et al.* [1,2]. In the stochastic version of this problem pioneered by Oommen [6,8,9], the LM attempts to locate a point in an interval with stochastic (i.e., possibly erroneous), instead of deterministic, responses from the Environment. Thus, when it should really be moving to the "right" it may be advised to move to the "left" and vice versa, with a nonzero probability.

Bentley and Yao [3] solved the deterministic point-location problem of searching in an unbounded space by examining points $f(i)$ and $f(i+1)$ at two successive iterations between which the unknown point lies, and by doing a binary search between them.

Unlike the work of Bentley and Yao [3], the faulty nature of the feedback from the Oracle in the SPL problem would render the latter task extremely intriguing and challenging. In fact, in [3], by virtue of the "correctness" nature of the feedback provided by the Oracle, the LM is able to confidently discard an entire interval from the search space each time it queries the Oracle. This approach would not be directly applicable to the SPL, as wrongly discarding a region that contains λ^* would mislead the LM. In this paper, we aim to design novel hierarchical learning schemes for solving the SPL problem that tolerate faulty feedback.

The main drawback of the state-of-the-art solution to the Stochastic-Point Location (SPL) problem reported in [6] is that the steps are always very conservative. If the step size is increased, the scheme converges faster, but the accuracy is correspondingly decreased. While if the step size is decreased the scheme yields a higher accuracy, but the convergence speed is correspondingly decreased.

In this paper we introduce the Hierarchical Stochastic Searching on the Line (HSSL) solution to the SPL problem. In the HSSL solution, as alluded to above, we structure the space as a binary tree. The algorithm then orchestrates a controlled RW on this space.

To informally clarify this, we preface our discussion by mentioning that for generations, the technique of searching a sorted list using "Binary Search" has been proven to outperform a controlled walk in the deterministic context. In the HSSL, this same principle has been extended to the SPL by discretizing the parameter space with a multilevel hierarchy, and performing the above-mentioned controlled RW on this space. The heart of the strategy involves performing controlled moves on a space structured as a tree, and then intelligently pruning the space using a hierarchical stochastic search. The rationale of the solution is to take advantage of the tree structure of the search space in order to enhance the search speed. This would enable the LM to quickly explore the search space and hopefully, focus its visits on the region that contains λ^*.

Apart from the problem being of importance in its own right, it also has potential applications in solving optimization problems. The details of this are omitted here, but can be found in[6].

2 Related Work

Oommen [6] proposed and analyzed an algorithm that operates on a discretized search space while interacting with an informative Environment ((i.e. $p > 0.5$). This algorithm takes advantage of the limited precision available in practical implementations to restrict the probability of choosing an action to only finitely many values from the interval $[0, 1]$, which also enables the analysis of the scheme.

The solution proposed in [6] for the SPL problem functions as follows. The space in which the search is conducted is first sliced by subdividing the unit interval into N steps $\{0, 1/N, 2/N, \ldots, (N-1)/N, 1\}$, and a larger value of N will ultimately imply a more accurate convergence to the unknown λ^*. The algorithm then invokes a controlled RW on this space. Whenever the LM is told to go to the right (left), it moves to the right (left) by a single step in this space.

The analytic results derived in [6] proved that if the "Oracle" was itself informative, the discretized RW learning was asymptotically optimal. Thus the mechanism would converge to a point arbitrarily close to the true value of λ^* with an arbitrarily high probability.

The primary drawback of the scheme described in [6] is the fact that the steps made are always very conservative. Thus, as stated above, if the step size is increased the scheme converges faster, but with a correspondingly less accuracy.

A novel strategy combining Learning Automata (LA) [5] and pruning was used in [8], which aims to search for the parameter in the continuous space

when interacting with an informative Environment. Utilizing the response from the Oracle, Oommen *et al.* partitioned the interval of search into three disjoint subintervals, eliminating at least one of the subintervals from further search and recursively searching the remaining interval(s) until the search interval is at least as small as the required resolution of estimation.

In a subsequent work [9], Oommen *et al.* introduced the Continuous Point Location with Adaptive d-ARY Search (CPL-AdS) which is a generalization of a portion of the work in [8]. In CPL-AdS, the given search interval is divided into d partitions representing d disjoint subintervals. In each interval, initially, the midpoint of the given interval was considered as the estimate of the unknown λ^*. Each of the d partitions of the interval is independently explored using an ϵ-optimal two-action LA, where the two actions are those of selecting a point from the left or right half of the partition under consideration. Then, the authors eliminated at least one of the subintervals from being searched further, and recursively searched the remaining pruned contiguous interval until the search interval is at least as small as the required resolution of estimation. This elimination process essentially utilizes the ϵ-optimality property of the underlying automata and the monotonicity of the intervals to guarantee the convergence. At each epoch consisting of a certain number, N_∞, of iterations, the algorithm "confidently" discarded regions of the search space.

In [7], Oommen *et al.* reported the first known solution to the stochastic point location (SPL) problem when the environment is non-stationary.

2.1 Contributions

Our paper presents a set of novel contributions summarized below:

- With regard to the design and analysis of discretized parameter schemes, we submit that a fundamental contribution of this paper is the manner in which we have designed the discretized search space, by structuring it as a balanced binary tree. Traditional approaches for discretization work by restricting the corresponding parameter to be one of finite number of values in the interval $[0, 1]$, and then a "one-dimensional" controlled RW is performed on the discretized space, where the transitions only occur between neighbor nodes, i.e to the "left" or to the "right". Instead, we propose a new philosophy for phenomenon of discretization *itself*, where the parameter space is structured as a binary tree. In brief, to each level of the tree, we associate a resolution that becomes finer at higher levels of the tree.
- The paper presents a significant contribution to the families of solutions relevant to the SPL problem.
- Extensive simulations results confirm that our scheme outperforms the state-of-the art discretized scheme [6]. We verify empirically that the proposed HSSL solution provides orders of magnitude faster convergence compared to the work reported in [6]. In addition, simulations results show that our scheme possesses an excellent ability to cope with nonstationary environments, both of which, we believe, are truly impressive!

– We report the first analytical results for HSSL and prove that the HSSL is asymptotically optimal. The analysis of the scheme is a contribution in its own right to the field of Markov Chains and LA.

3 Merging the Fields of Binary Search and SPL in HSSL

The algorithm we propose operates by invoking a controlled RW on a tree. The space of the search is arranged in the form of a binary tree with depth $D = log_2(N)$, where N, (which, for the sake of simplicity, is assumed to be a power of 2) is the resolution of the algorithm. The LM searches for the optimal value λ^* by traversing the tree, moving from one tree node to another. The way by which this is achieved in explained below.

3.1 Definitions

Construction of Hierarchy: Let $\Delta = [\sigma, \gamma)$ be the current search interval containing λ^* whose left and right (smaller and greater) boundaries on the real line are σ and γ, respectively. Without loss of generality, we initially assume that $\sigma = 0$ and $\gamma = 1$. The search space is constructed as follows: First of all, the hierarchy is organized as a balanced binary tree with depth D. To each node in the hierarchy we associate an interval. For convenience, we will index the nodes using their depth and their relative order with respect to the nodes situated at the same the depth.

Root Node: The root of the hierarchy (at depth 0), which we call $S_{\{0,1\}}$, is assigned the interval $\Delta = \Delta_{\{0,1\}} = [0, 1)$.

This interval is partitioned into two disjoint equisized[1] intervals $\Delta_{\{1,1\}}$ and $\Delta_{\{1,2\}}$, such that $\Delta_{1,1} = [0, 1/2)$ and $\Delta_{1,2} = [1/2, 1)$. Note that $1/2 = mid(\Delta_{\{0,1\}})$, where $mid(\Delta_{\{0,1\}})$ denotes the midpoint of $\Delta_{\{0,1\}}$. To avoid confusion, we shall use the notation[2] that refers to the interval $\Delta_{\{1,1\}}$ as the *Left Child* of the root and to $\Delta_{\{1,2\}}$ as its *Right Child*.

Nodes at depth d: The node $j \in \{1, ..., 2^d\}$ at depth d, is referred to as $S_{\{d,j\}}$ for $1 < d < D$. This node is assigned the interval $\Delta_{\{d,j\}} = [\sigma_{\{d,j\}}, \gamma_{\{d,j\}})$, which is associated with two disjoint equisized intervals $\Delta_{\{d+1,2j-1\}}$ and $\Delta_{\{d+1,2j\}}$.

Following the same previously alluded to nomenclature, $\Delta_{\{d+1,2j-1\}}$ is the *Left Child* of $\Delta_{\{d,j\}}$, and $\Delta_{\{d+1,2j\}}$ is its *Right Child*.

Nodes at depth D: At depth D, which represents the maximal depth of the tree, the nodes do not have children. In fact, when the search interval is at least as small as the required resolution of estimation, we do no further partitioning.

By virtue of the equi-partitioning property, for a given node j at depth d that is associated with the respective interval $\Delta_{\{d,j\}}$, we can deduce the values

[1] The equi-partioning is really not a restriction. It can be easily generalized.

[2] Indeed, we shall utilize the notations that *Parent*, *Left Child* and *Right Child* of an interval $\Delta_{\{i,j\}}$ in the binary tree are the intervals associated to the respective *Parent*, *Left Child* and *Right Child* of the node $S_{\{i,j\}}$.

of the left and right boundaries of the interval to be: $\sigma_{\{d,j\}} = (j-1)(\frac{1}{2})^d$ and $\gamma_{d,j} = j(\frac{1}{2})^d$, for $j \in \{1, ..., 2^d\}$ where $1 \leq d \leq D$.

Boundary Value Convention Regarding Notation: Since level "-1" is nonexistent, we use the notation appropriate for the boundary (basis case) condition, and denote the *Parent* of $\Delta_{\{0,1\}}$ to be $\Delta_{\{0,1\}}$ itself. The same comment is also valid for the root node. In other words: $Parent(\Delta_{\{0,1\}})=\Delta_{\{0,1\}}$.

Also, since level $D+1$ is nonexistent, we use the convention that the *Right Child* of a leaf node is the same as the leaf node itself. Similarly, the *Left Child* of a leaf node is the leaf node itself. Thus, formally, we call $Left\ Child(\Delta_{\{D,j\}})=Right\ Child(\Delta_{\{D,j\}})=\Delta_{\{D,j\}}$ for $j \in \{1, ..., 2^D\}$. The same notation applies as well to the leaf nodes $S_{\{D,j\}}$.

Target Node: We define the **Target** node as the leaf node whose associated interval contains λ^*.

Non-Target Nodes: The **Non-Target** nodes are leaf nodes whose corresponding associated intervals do not contain λ^*.

Resolution: Whenever the LM is at a certain node in the tree, we propose to use, as an estimate of the unknown λ^*, the middle point of the interval itself. By virtue of the equi-partitioning of the intervals at each level of the tree, whenever the LM is at node of a certain depth d in the tree, the estimate of the unknown λ^* will take a discretized value, a multiple of $(\frac{1}{2})^{d+1}$.

We call the resolution of the scheme, the number of leaf nodes, i.e $N = 2^D$.

3.2 Structure of the Search Space and Responses from the Environment

We intend to organize the search space in the form of a balanced binary tree, where each node corresponds to an interval range. Initially, we guess the midpoint of the given interval to be our estimate of the unknown λ^*. The LM searches for the optimal value λ^* by operating a RW on the tree, moving from one tree node to another, with the goal of locating the *target leaf node*. Each node in the tree is associated with an interval; e.g., the root is associated with the interval $[0, 1)$. This interval is partitioned into two disjoint equi-sized intervals. Thus, the left child of the root is associated with $[0, 1/2)$, the right child with $[1/2, 1)$, etc.

At any given time instance, let us assume that the LM finds itself at a node $S_{\{d,j\}}$ in the tree, where $j \in \{1, ..., 2^d\}$ and $1 \leq d \leq D$. The LM attempts to infer the next promising search interval that is likely to contain λ^* by making a sequence of "informed" guesses. For each guess, the Environment essentially informs the LM, possibly erroneously (i.e., with probability p), which way it should move to reach the unknown point. Let $\Delta_{\{d,j\}}$ be the interval that is associated with the node where the LM resides at the current time instant. The informed guesses correspond to a sampling at the boundary points of the interval: $\Delta_{\{d,j\}}$, *and* at the midpoint of the interval: $mid(\Delta_{\{d,j\}})$.

We formalize this by saying that the sampled points can be expressed as a vector $\overrightarrow{x} = [x^1, x^2, x^3]$, where $x^1 = \sigma_{\{d,j\}} = (j-1)(\frac{1}{2})^d$, $x^2 = mid(\Delta_{\{d,j\}}) = (2j-1)(\frac{1}{2})^{d+1}$ and $x^3 = \gamma_{\{d,j\}} = j(\frac{1}{2})^d$.

Further, let the corresponding respective responses from the Environment E be formulated as a tuple: $\overrightarrow{\Omega} = [\Omega^1, \Omega^2, \Omega^3]$.

Note that Ω^k, for $k \in \{1, 2, 3\}$, is a random variable that can take either the value Left or Right. Since the environment is assumed faulty, we suppose that it suggests the correct direction with a probability p. Therefore Ω^k, for $k \in \{1, 2, 3\}$ can be formally defined according to whether λ is bigger or smaller than x^k as:

If $\lambda < x^k$

$$\Omega^k = \begin{cases} L & \text{with probability } p \\ R & \text{with probability } (1-p), \text{and} \end{cases}$$

If $\lambda \geq x^k$

$$\Omega^k = \begin{cases} L & \text{with probability } (1-p) \\ R & \text{with probability } p, \end{cases}$$

where for simplicity, we use L to imply a region to the "left" of the sampled point, and R to imply a point to the "right" of the sampled point.

As a consequence of the above, it is easy to see that the overall effect of the Environment E is that it responds with one of the 2^3 possible results:

{[L, L, L], [L, L, R], [L, R, L], [L, R, R], [R, L, L], [R, L, R], [R, R, L], [R, R, L]}.

Based on these responses the LM moves to another node in the tree, either to the current node's parent, or to one of its children (*Left Child*/*Right Child*). The rules for moving in the tree are summarized in Table 1, and the formal result about the algorithm is stated in Theorem 1.

Table 1. Decision table to choose the next search interval based on the response vector $[\Omega^1, \Omega^2, \Omega^3]$, when the current search interval is $\Delta_{\{i,j\}}$

Next Search Interval	**Condition**
$Parent(\Delta_{\{i,j\}})$	[R, R, R] $\bigvee$ [L, R, R] $\bigvee$ [L, L, R] $\bigvee$ [L, L, L]
$LeftChild(\Delta_{\{i,j\}})$	[R, L, R] $\bigvee$ [R, L, L]
$RightChild(\Delta_{\{i,j\}})$	[R, R, L] $\bigvee$ [L, R, L]

Theorem 1: The parameter learning algorithm specified by the rules summarized in Table 1 is asymptotically optimal if p is bigger than the conjugate of the golden ratio[3]. Formally, $Lim_{N\to\infty} Lim_{n\to\infty} E[\lambda(n)] \to \lambda^*$.

In the interest of brevity, we omit the proof of Theorem 1. The proof is quite involved and can be found in [10]. □

[3] The golden ratio conjugate quantity is defined as $\Phi = \frac{\sqrt{5}-1}{2}$.

4 Simulation Results

In this section, we present the results which demonstrate the power of the HSSL for the SPL. In order to confirm the superiority of our scheme, we have conducted extensive simulations results under different parameter settings. However, in the interest of brevity, we merely cite a few specific experimental results.

In Table 4 we have recorded the true value of $E[\lambda(\infty)]$ for various values of p and the tree depth $D = log_2(N)$ (i.e, the resolution is $N = 2^D$) when the value of λ^* is 0.9123. The values of p are 0.7, 0.85, and 0.95.

The optimality property is empirically confirmed through the simulation. Observe that independent of whether the value of p is as low as 0.7 or as high as 0.95, $E[\lambda(\infty)]$ indefinitely approaches the optimal λ^* as we increase the resolution. Note that for a depth D which equals 12, the final terminal value represents an error less than 0.0005% for $p = 0.7$, $p = 0.85$, and $p = 0.95$.

We now report the results of the second set of experiments in which we have tried to catalogue the convergence of $E[\lambda(n)]$ with time, "n".

In order to obtain an understanding as to how the scheme converges with time, various simulations were conducted to evaluate the performance of the algorithm under a variety of constraints. In each simulation, 100 parallel experiments were conducted so that an accurate ensemble average of the results could be obtained. Again, although numerous experiments have been conducted, in the interest of brevity we shall merely report the results obtained for one set of experiments involving the unknown parameter λ^* switching periodically between the values 0.9123 and $1-0.9123$. We compared our results to the algorithm presented in [6]. Also, to be on the same level playing field, since there was no *apriori* information about the value λ^*, at time instant 0, we initialized the LM of the original SPL scheme to the position $\frac{N}{2}$, while the corresponding LM of the HSSL algorithm started from the root node.

In order to understand the effect of the resolution on the rate of convergence, we report the number of iterations required to reach a value that is 95% of the final value of λ^*.

In all brevity, we state that the HSSL algorithm outperformed the original SPL solution and learned the value of λ^* much faster. The experimental results obtained are truly conclusive.

In the first set of experiment, we fixed p to 0.8. The plots of the corresponding results are shown in Fig. 1(a) and Fig. 1(b). In Fig. 1(a), the resolution N was equal to 256 while λ^* switched every 400^{th} iteration. In Fig. 1(b), the resolution N was equal to $1,024$, while λ^* switched every $1,500^{th}$ iterations.

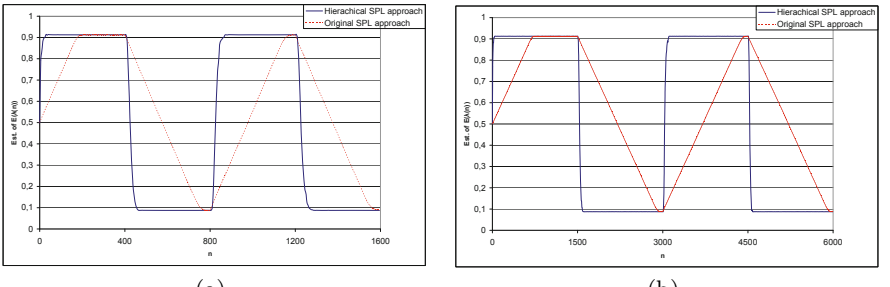
From Fig. 1(a), we see that in the first 400 iterations, it took the HSSL solution only 30 time instants to reach 95% of λ^*, while the original SPL solution required 180 iterations. After the first environment "switch", i.e between time instants 400 and 800, we observe that the convergence speed of both algorithms decreased slightly. In fact, 95% of λ^* was attained within 45 subsequent iterations in the case of the HSSL solution, while the original SPL solution took 350 iterations. Comparing the results of the first 400 iterations with that of the subsequent iterations, we conclude that although the final steady-state probabilities

are independent of the starting state, in reality, the time that the LM took to converge to λ^* is dependent on where one starts.

In Fig. 1(b), we show the results when we increased the resolution N to $1,024$. As in the previous case, from Fig. 1(b), we observe that in the first $1,500$ iterations, it took approximately only 50 iterations for our HSSL solution to reach 95% of the optimal value λ^*, while the original SPL solution spent 680 iterations. After the first environment switch, i.e between time instants 1500 and 3000, we observe that the convergence speed decreased. In fact, it took approximately 75 iterations for our HSSL solution to reach 95% of the optimal value λ^*, while the original SPL solution required $1,380$ iterations. In these settings, the HSSL approach provided an order of magnitude (i.e., 18) faster convergence than the original SPL solution. This, we believe, is impressive. We further remark that, as we increased the resolution N from 256 (see Fig. 1(a)) to 1024 (see Fig. 1(b)) for the same $p = 0.8$, the convergence speed of the original SPL solution was significantly reduced while the speed of the HSSL was less affected by this.

Table 2. True value of $E[\lambda(\infty)]$ for various values of p and various resolutions, when the value of λ^* is 0.9123

$log_2(N)$	$p = 0.7$	$p = 0.85$	$p = 0.95$
2	0.727906125	0.825522	0.866370875
3	0.82995875	0.9204060625	0.9337756875
4	0.86781290625	0.90972528125	0.9076381875
5	0.89240075	0.91925609375	0.921083703125
6	0.9056527421875	0.9150871328125	0.9144455
7	0.9069764765625	0.91111155859375	0.91035579296875
8	0.908265740234375	0.911884287109375	0.912022787109375
9	0.911209423828125	0.9128465126953125	0.9130258076171875
10	0.9116439086914062	0.912655470703125	0.9126211733398437
11	0.91219955078125	0.9124482912597657	0.912371505859375
12	0.9121167028808593	0.9122652081298828	0.9122371402587891



(a) (b)

Fig. 1. The learning characterstics of the HSSL and the original SPL algorithm when λ^* switches between the values 0.9123 and $1.0-0.9123$ every: (a) 400^{th} iterations where $N = 256$ and $p = 0.8$, and (b) 1500^{th} iterations where $N = 1024$ and $p = 0.8$

5 Conclusions

The SPL problem involves a LM that attempts to learn a parameter, for example, λ^*, within a closed interval. For each guess, the environment essentially informs the mechanism with three responses, each possibly erroneous (i.e., with probability p), on which way it should move to reach the unknown point. We have presented a solution that involves discretizing the space, mapping the discretized intervals onto a binary tree and performing a controlled random walk on this space. The solution we have presented has been both analytically analyzed and simulated, with extremely fascinating results. Apart from formally analyzing this algorithm, we have also experimentally demonstrated its superiority over the state-of-the-art. From this perspective, our approach has been shown to provide orders of magnitude faster convergence speed than the traditional SPL solution [6] in non-stationary environments, i.e., where λ^* changes over time.

References

1. Baeza-Yates, R., Culberson, J., Rawlins, G.: Searching with uncertainty. In: Proceedings of Scandinavian Workshop Algorithms and Theory (SWAT), Halmstad, Sweden, pp. 176–189 (1998)
2. Baeza-Yates, R., Schott, R.: Parallel searching in the plane. Comput. Geom. Theory Appl. 5, 143–154 (1995)
3. Bentley, J.L., Yao, A.C.C.: An Almost Optimal Algorithm for Unbounded Searching. Inform. Proc. Lett. 5, 82–87 (1976)
4. Livio, M.: The Golden Ratio: The Story of Phi, the World's Most Astonishing Number. Paw Prints (2008)
5. Narendra, K.S., Thathachar, M.A.L.: Learning Automata: An Introduction. Prentice-Hall, Inc. (1989), http://portal.acm.org/citation.cfm?id=64802
6. Oommen, B.J.: Stochastic searching on the line and its applications to parameter learning in nonlinear optimization. IEEE Transactions on Systems, Man and Cybernetics 27B, 733–739 (1997)
7. Oommen, B.J., Kim, S.W., Samuel, M., Granmo, O.C.: A solution to the stochastic point location problem in metalevel nonstationary environments. IEEE Transactions on Systems, Man, and Cybernetics 38(2), 466–476 (2008)
8. Oommen, B.J., Raghunath, G.: Automata learning and intelligent tertiary searching for stochastic point location. IEEE Transactions on Systems, Man and Cybernetics 28B, 947–954 (1998)
9. Oommen, B.J., Raghunath, G., Kuipers, B.: Parameter learning from stochastic teachers and stochastic compulsive liars. IEEE Transactions on Systems, Man and Cybernetics 36B, 820–836 (2006)
10. Yazidi, A., Granmo, O.C., Oommen, B.J., Goodwin, M.: Hierarchical stochastic searching on the line. Unabridged version of this paper (to be submitted for publication)